Predicting Genotypes from Allele Intensity Data Using a Support Vector Machine

Daan Leiva
UCLA
daanleiva@g.ucla.edu

***Abstract*— A important initial step in improving a prediction system is being able to duplicate the results of the current model to verify that your prediction model is functioning properly. In this preliminary work, we will find the optimal SVM model type and its corresponding parameters for analyzing consistent genotype data with at least one instance of MCSS $\leq$ 5.**

I. INTRODUCTION

To correctly predict genotypes from allele frequencies, companies who focus in SNP genotyping must gather redundant data to achieve high levels of accuracy and reliability. The extra processes and analysis that take place as a result can lead to high overhead costs. One method of minimizing these costs is to improve the accuracy of the machine learning algorithms which process the allele frequency data. These improvements can lead to a reduction in materials cost, computational time and an increase in the degree of confidence in the results. One such technology used for SNP genotyping is the OpenArray™ which allows for high-throughput of single nucleotide polymorphisms (SNPs) genotypes. In this project, we focused on analyzing the large data set provided by the OpenAray™ technology and selecting the best model for the predicting of two-allele frequency data using a Support Vector Machine (SVM) learning system.

II. BACKGROUND AND RELATED WORK

A. Support Vector Machine

Support Vector Machines are a supervised learning technique used to classify linearly separable patterns (Fig. 1). Patterns that are not linearly separable can be mapped by a kernel function into a linearly separable space (Fig. 2).

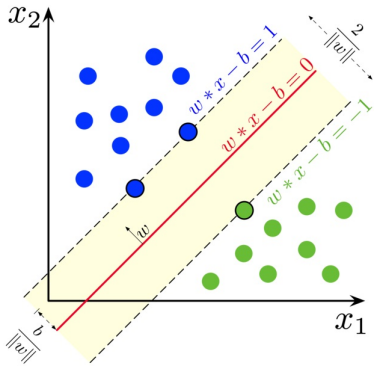

Fig. 1: SVM Margin

Two of the more common kernels are a linear function and a radial basis function (RBF). The linear kernel proposes a simple transformation and its optimization function can be calculated relatively fast. The RBF or Gaussian transformation focuses on the distance a data point has from an origin to create a linearly separable space. While an RBF transformation can often produce more accurate results, it also has a significantly longer training period due to the higher complexity of its optimization function. The training algorithms for both functions have parameters that need to be optimized.

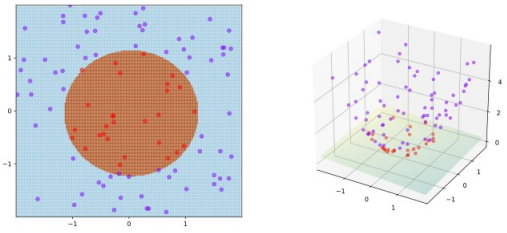

Fig. 2: RBF Kernel

A common parameter to both models is the penalty parameter. The penalty parameter is tied to the weight of an error term. A low penalty value can lead to a simple boundary surface, while a high penalty value focuses on classifying training examples accurately. The RBF model also includes a gamma parameter. The gamma parameter controls the influence of each single training example has. A high gamma value leads to a smaller effect radius.

B. OpenArray™

The data for this project was gathered by OpenArray™ technology. Two allele specific TaqMan probes are used for capturing genetic data[1]. In order to detect specific SNPs the probes use a PCR primer pair and fluorescent dyes. This process generates intensity data that can be translate to a genotype (Fig. 3). The two intensity values acquired are VIC and FAM[2]. Allele 1 is tied to VIC, while allele 2 is tied to FAM. Figure 4 demonstrates some linear separability and is one of the primary reasons we chose the SVM as our training model. The dye intensities classify the genotypes in the following way: a high intensity for VIC points to a homozygous genotype for allele 1, while a high intensity for FAM points to a homozygous genotype for allele 2. If both intensities are presented with similar strengths, we classify the sample as heterozygous genotype.

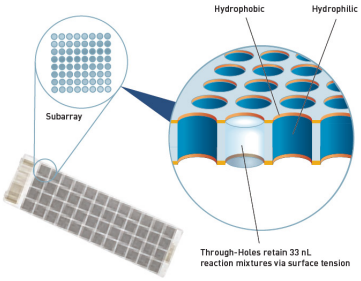

Fig. 3: OpenArray™

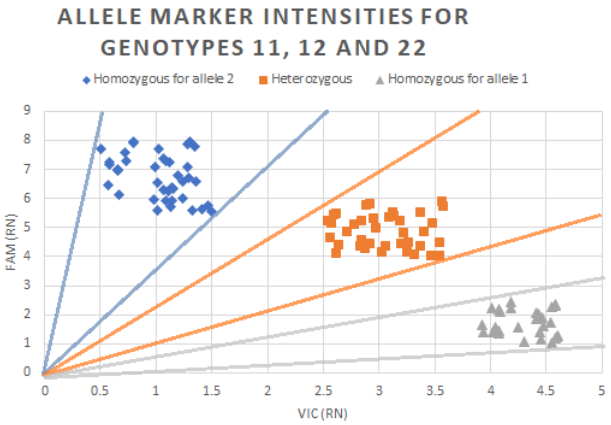

Fig. 4: Allele Marker Intensity Separability

C. Quality Metrics

A Minimum Cluster Sigma Separation (MCSS) score is assigned to each assay (a collection of repeated measurements). The MCSS score quantifies how clearly separated the allele intensities are. For example, Figure 4 demonstrates a case with a high MCSS score since all the plotted allele intensities can be easily assigned to a genotype cluster. An MCSS of less than 5 leads to a discarded analysis. When a single analysis is discarded, the entire assay is marked as a failed assay. Recovering failed assays is one of the main goals of this project since failed assays represent about 7.5% of the total data.

D. Programming Languages

Two programming languages were used in this project: R and python. Rs statistical analysis toolkit was used for processing data as well as transferring it into our database. R studios grid view allowed us to preview our transformation outputs and helped in the debugging process. Python and its comprehensive scikit-learn library were used for training our SVM as well as testing our predictor. We decided to use SQLite as our database management system since it is open source and compatible with both, Linux and Window systems.

1) Scikit-Learning: This machine learning library for Python supports multiple algorithms to handle a variety of machine learning tasks[3] such as clustering, regression, dimensionality reduction and classification. Some of the learning models included are SVM, K-means and Lasso regression. Scikit-learn reduces the programming overhead work by providing an interface that only requires the user to select a model and parameter. Scikit-learn will be used through our project to train and predict our SVM model.

2) Pandas: Pandas is an open source python library created for data structure manipulation and management.

III. Methods and Implementation

A. Working Data Set

Our data pool consisted of 8,097 SNP assays. From this data pool 3,506 assays had at least one instance of MCSS $\leq$ and 4,591 assays without any instances of MCSS $\leq$ 5. This project focuses on the 3,506 assays that had at least one instance of MCSS $\leq$ 5 (failed essays) since the lack of separation makes them harder to predict. This group of assays can be further separated into a discrepant and consistent category. The consistent assays predict the same genotype. 92.5% of these failed-assays belong to the consistent category (3,243) and only 7.5%-symbol (263) of the failed assays belong to the discrepant category. In this preliminary work, we focus on the 263 discrepant assays. Lastly, we decided to ignore assays that were labeled as Low-ROX (this is directly tied to the dye florescence intensity) since this quality also demonstrates a low confidence in the measurements.

As explained below, we decided to use a 70% majority rule to label our data set. This means that if within an assay at least 70% of its data points belonged to the same genotype, then we assign that genotype to be the ground truth for the assay. To train and test our model on both, consistent and inconsistent arrays, we randomly selected 500,000 arrays from the consistent category. Combining our inconsistent and consistent data points yields 939,643 data points. These data points were split 75/25 into a training and testing data sets with equal ratios of consistent to inconsistent data points.

TABLE I: Assay Distribution by Category

	Consistent	Discrepant	Total
At Least One MCCS <5	3243	263	3506
No MCCS <5	4511	61	4572
Total	7754	324	8078

B. Majority Threshold for Ground Truth

In order to label discrepant assays, we came up with a majority threshold rule that can define the ground truth. This means that if the assigned percentage of arrays within an assay belong to the same category, then we can label that assay by the majority genotype and call it the ground truth for those data points. Since our data set is primarily composed of consistent data (25,000,000) when varying the majority threshold percentage, we focused on how many inconsistent data points this provided us. As expected, as the majority threshold is increased, the number of arrays that qualify decreases. The results for varying the majority threshold can be seen in the table below. In order to have a large enough training pool while maintaining a high enough majority

threshold, we decided to use 70% as our majority threshold. This provided 439,643 data points which represent roughly 86% of all the arrays within the inconsistent category.

TABLE II: Majority Threshold Counts for Inconsistent Arrays

Majority Threshold	Number of Arrays Above Threshold	Percent of Total Arrays
90%	350,261	68.16%
80%	401,908	78.21%
70%	439,643	85.55%
60%	485,717	94.52%

C. Python Code for Training and Evaluation

Our training and testing program were written in Python. Simplified versions of the primary methods used are described below.

1) Main Function: The main function handles the overhead work of loading the locations of the testing and training files. It also calls the training function and prediction function.

```
def main(parser):
  option = parser.parse_args()
  # load file locations
  train_file = option.train
  test_file = option.test

  # train models
  (models, train_acc, norm_model) = train_svm_model
    (train_file)

  # evaluate models
  test_evaluation = svm_prediction(test_file,
    models, norm_model)
```

Listing 1: Main

2) Loading Data: The load matrix function reads one of the two data files (train or test) and parses it into an X matrix and y vector. The X matrix will contain our two key features (allele 1 and allele 2 intensities) while the y vector will contain the true genotypes used for training and evaluating.

```
def load_matrix(filename):
  X = []
  y = []

  fh = open(filename, 'rb')
  title = fh.readline()
  for line in fh:
    mini_data = line.rstrip().split("\t")
    allele_1 = float(mini_data[3])
    allele_2 = float(mini_data[4])
    true_genotype = mini_data[6].rstrip('\"').
    lstrip('\"')

    X.append( [allele_1, allele_2])
    y.append(true_genotype)

  fh.close()

  return  (X,y)
```

Listing 2: Load Matrix

3) Training Method: The training method is in charge of fitting our models using the training data. It also needs to return the normalized model so that the test data can be normalized by the same factor during our evaluation phase. The training function also iterates through our parameter values and generates a model for each combination of parameters. In the case of the linear SVM the only parameter we modify is the penalty factor c, for our RBF model we also need to iterate through the kernel coefficient g.

```
def train_svm_model(train_file):

  # Load training data
  (X,y) = load_matrix(train_file)

  # Normalizes data
  norm_model = MinMaxScaler()
  norm_X = norm_model.fit_transform(X)

  # Train Models
    C_range = [0.01, 0.03, 0.1, 0.3, 1, 3, 10]
  fits = []
  acc = []
  for c in C_range:
    # fitted model
    f = svm.SVC(kernel='linear', C = c)

        fits.append(f.fit(norm_X, y))
        acc.append(f.score(norm_X, y))

  return (fits, acc, norm_model)
```

Listing 3: Train Linear SVM

```
def train_svm_model(train_file):

  # Load training data
  (X,y) = load_matrix(train_file)

  # Normalizes data
  norm_model = MinMaxScaler()
  norm_X = norm_model.fit_transform(X)

  # Train Models
    C_range = [0.1, 0.3, 1, 3, 10, 30, 100]
    g_range = [1, 3, 10, 30, 100, 300, 1000]
  fits = []
  acc = []
  for c in C_range:
      for g in g_range:
          # fitted model
        f = svm.SVC(kernel='linear', C = c, gamma =
    g)

            fits.append(f.fit(norm_X, y))
            acc.append(f.score(norm_X, y))

  return (fits, acc, norm_model)
```

Listing 4: Train RBF SVM

4) Evaluate Method: The evaluate method is in charge of calculating the multiple metrics that help us qualify our models. The classification_report function of the scikit-learn package can produce the precision, recall and F-1 scores. We also use the score function to calculate the overall accuracy. Lastly, this method also generates a prediction matrix to analyze how well each genotype was predicted.

```
def svm_prediction(test_file, models, norm_model):
  echo("Loading Matrix")
```

```
  (X, y) = load_matrix(test_file)

  echo("Normalization")
  norm_X = norm_model.fit_transform(X)

  evaluation = []

  for m in models:
    predict_y = m.predict(norm_X)

    # compute overall accuracy
    acc = m.score(norm_X, y)

    # classification report
    target_names = ["11", "12", "22"]
    report = classification_report(y, predict_y,
    target_names = target_names)

    # prediction matrix
    prediction_matrix = [[0,0,0],[0,0,0],[0,0,0]]

    fh = open("svm_linear_minmax.txt", 'wb')
    fh.write("Prediction\tTruth\n")

    for i in xrange(len(predict_y)):

      fh.write("%s\t%s\n" %(predict_y[i], y[i]))

      if(predict_y[i] == "11" and y[i] == "11"):
        prediction_matrix[0][0] += 1
      elif(predict_y[i] == "11" and y[i] == "12"):
        prediction_matrix[0][1] += 1
      elif(predict_y[i] == "11" and y[i] == "22"):
        prediction_matrix[0][2] += 1
      elif(predict_y[i] == "12" and y[i] == "11"):
        prediction_matrix[1][0] += 1
      elif(predict_y[i] == "12" and y[i] == "12"):
        prediction_matrix[1][1] += 1
      elif(predict_y[i] == "12" and y[i] == "22"):
        prediction_matrix[1][2] += 1
      elif(predict_y[i] == "22" and y[i] == "11"):
        prediction_matrix[2][0] += 1
      elif(predict_y[i] == "22" and y[i] == "12"):
        prediction_matrix[2][1] += 1
      elif(predict_y[i] == "22" and y[i] == "22"):
        prediction_matrix[2][2] += 1

    fh.close()
    evaluation.append( (acc, report,
    prediction_matrix))
  return evaluation
```

Listing 5: Evaluate

IV. Evaluation

A. Model Selection

We considered two models for our analysis: a Linear SVM and a RBF SVM. Their accuracy is laid out in the tables below. The linear SVM showed little sensitivity to the parameter sweep with its testing accuracy remaining the same. The RBF presented higher sensitivity to parameter tuning and it also produced a higher accuracy within the same range of the penalty parameter c. However, the linear model training time was significantly faster. The training time was less than a day for the Linear models while the RBF model with high g and c values took a couple of days to finish.

1) Linear SVM: The Linear SVM model only showed an absolute range of 0.01% in the overall accuracy (Table III).

TABLE III: Results for the Linear SVM

c	Training Accuracy	Testing Accuracy	F-1 Score	Overall Accuracy
0.01	0.7324	0.7337	0.71	0.732725
0.03	0.7324	0.7338	0.71	0.73275
0.1	0.7324	0.7338	0.71	0.73275
0.3	0.7325	0.7338	0.71	0.732825
1	0.7325	0.7338	0.71	0.732825
3	0.7324	0.7338	0.71	0.73275
10	0.7325	0.7338	0.71	0.732825
30	0.7325	0.7338	0.71	0.732825

2) RBF SVM: The RBF SVM model revealed an absolute range of 0.29% for its overall accuracy and its least accurate model outperformed the best Linear SVM model (Table IV).

TABLE IV: Results for the RBF SVM

c	g	Training Accuracy	Testing Accuracy	F-1 Score	Overall Accuracy
0.1	1	0.742109	0.743264	0.72	0.74239775
0.1	3	0.743044	0.744039	0.73	0.74329275
0.1	10	0.744333	0.745287	0.73	0.7445715
0.3	1	0.742279	0.743294	0.72	0.74253275
0.3	3	0.74317	0.744261	0.73	0.74344275
0.3	10	0.745105	0.745755	0.73	0.7452675
1	1	0.742394	0.74329	0.72	0.742618
1	3	0.743412	0.744346	0.73	0.7436455
1	10	0.744794	0.745453	0.73	0.74495875
3	1	0.742416	0.743354	0.72	0.7426505
3	3	0.74362	0.744656	0.73	0.743879
3	10	0.744934	0.745453	0.73	0.74506375
10	1	0.7426	0.743614	0.72	0.7428535
10	3	0.74393	0.74495	0.73	0.744185
10	10	0.745106	0.745499	0.73	0.74520425

B. RBF Parameter Sweep

Since the RBF model produced an overall higher accuracy compared to the Linear SVM model, we chose it for further analysis. In this section we present the results of a parameter sweep on the RBF model. The range of the penalty value c values was from 0.1 to 100 and the kernel coefficient from 1 to 1000.

The highest accuracy was produced by model c = 1 and g = 1000, however we chose model c = 0.3 and g = 300 for further analysis, since the higher c and g value models significantly increased the training time for our model. Figure V shows that both points reside near the peak of the accuracy plot. With only a minor sacrifice in accuracy, our second best model training time was reduced from a couple of days to about a day.

TABLE V: Parameter Sweep Results for RBF SVM Models

c	g	Training Accuracy	Testing Accuracy	F-1 Score	Overall Accuracy
0.1	1	0.742109	0.743264	0.72	0.74239775
0.1	3	0.743044	0.744039	0.73	0.74329275
0.1	10	0.744333	0.745287	0.73	0.7445715
0.1	30	0.74512	0.745708	0.73	0.745267
0.1	100	0.74594	0.7463	0.73	0.74603
0.1	300	0.746712	0.746244	0.73	0.746595
0.1	1000	0.746779	0.744546	0.73	0.74622075
0.3	1	0.742279	0.743294	0.72	0.74253275
0.3	3	0.74317	0.744261	0.73	0.74344275
0.3	10	0.745105	0.745755	0.73	0.7452675
0.3	30	0.745322	0.745802	0.73	0.745442
0.3	100	0.746324	0.746474	0.73	0.7463615
0.3	300	0.747576	0.746828	0.73	0.747389
0.3	1000	0.749644	0.746338	0.73	0.7488175
1	1	0.742394	0.74329	0.72	0.742618
1	3	0.743412	0.744346	0.73	0.7436455
1	10	0.744794	0.745453	0.73	0.74495875
1	30	0.745527	0.745853	0.73	0.7456085
1	100	0.746539	0.746457	0.73	0.7465185
1	300	0.748065	0.746798	0.73	0.74774825
1	1000	0.750949	0.746942	0.73	0.74994725
3	1	0.742416	0.743354	0.72	0.7426505
3	3	0.74362	0.744656	0.73	0.743879
3	10	0.744934	0.745453	0.73	0.74506375
3	30	0.745654	0.745921	0.73	0.74572075
3	100	0.746834	0.746478	0.73	0.746745
10	1	0.7426	0.743614	0.72	0.7428535
10	3	0.74393	0.74495	0.73	0.744185
10	10	0.745106	0.745499	0.73	0.74520425
30	1	0.742729	0.743643	0.72	0.7429575
30	3	0.74468	0.745312	0.73	0.744838
100	1	0.742938	0.743822	0.72	0.743159
100	3	0.744673	0.745257	0.73	0.744819

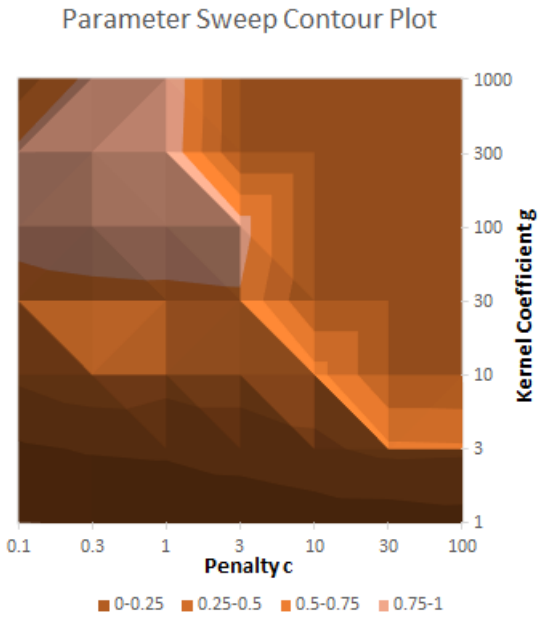

Fig. 5: RBF Contour Plot

C. Prediction Results with Best RBF Model

Table VI summarizes the results produced by evaluating our chosen model (c = 0.3 and g = 300) on our test set. Predicting the genotype 12 has the lowest performance in every statistical category. This was expected due to genotype 12 having the least amount of linear separation from the other two types.

Table VI summarizes four statistically significant figures:

TABLE VI: Test Set Results for Model g = 0.3 and g = 300

	Precision	Recall	F-1 Score	Support
11	0.76	0.90	0.82	100,993
12	0.62	0.33	0.43	43,154
22	0.77	0.77	0.77	90,761
Average / Total	0.74	0.75	0.73	234,911

precision, recall, F-1 score and support for each genotype in our experiment. The precision is a measure of how many of our positive predictions were actually correct, the recall refers to our true positive rate or the percentage of positive cases that we manage to capture, the F1-score is a mixed measure of both the recall and precision. Lastly, support refers to the quantity of true cases for that class. Below are the formulas for each statistical definition in terms of the true positive (tp), false positive (fp), true negative (tn), and false negative (fn) counts.

$$precision = \frac{tp}{tp + fp} \tag{1}$$

$$recall = \frac{tp}{tp + fn} \tag{2}$$

$$F-1Score = 2 \cdot \frac{recall \cdot precision}{recall + precision} \tag{3}$$

Table VII gives a breakdown of the prediction rate cross referenced by genotype for our test set. By looking at the total for the true counts we can see that genotype 12 has a lower count compared to genotype 11 and 22.

TABLE VII: Prediction Table for Model g = 0.3 and g = 300

	True 11	True 12	True 22	Total
Predicted 11	91,061	14,431	15,067	120,559
Predicted 12	3,255	14,371	5,684	23,310
Predicted 22	6,677	14,352	70,010	91,039
Total	100,993	43,154	90,761	234,908

D. K-Fold Validation

To test how well our model generalizes to different data sets we applied a 4-fold cross-validation[4]. We randomly partitioned our data into 4 sets. We then used three sets for training and one set for testing. The results for each fold are shown in Table VIII. Our results show that our model generalizes well to each randomly generated fold.

E. Predicting Non-Majority Genotypes

To qualify the predicting capabilities of our model further we will measure the cohesiveness of our predictions when working with non-majority data. As described in the Working Data Set section the method we used to label our data was to only label those data sets where the arrays within a sample

TABLE VIII: K-Fold Validation for Model g = 0.3 and g = 300

	Training Accuracy	Testing Accuracy	Overall Accuracy	F-1 Score
k-1	0.747595	0.746781	0.7473915	0.73
k-2	0.747965	0.745865	0.74744	0.73
k-3	0.747907	0.745886	0.74740175	0.73
k-4	0.747576	0.746845	0.74739325	0.73

had a majority genotype above 70%. In this section we will try to predict data that did not meet that 70% majority threshold and we will analyze how well our model can predict a majority genotype. We can see from our results

TABLE IX: Predicting Accuracy for Non-Majority Data

Majority Threshold	# of samples above majority threshold	# of samples	Ratio
0.6	339	420	0.807
0.7	279	420	0.664
0.8	204	420	0.486
0.9	123	420	0.293
0.95	80	420	0.19

that even though none of these samples achieved a majority genotype as per their original labels, our model managed to achieve a much higher cohesiveness. Even at a higher threshold than we set originally such as 80% we can see that roughly half of the samples have a majority genotype. These results do not show the accuracy of our results but instead qualify how well our model can predict data from the same samples.

V. Conclusions

In this work we managed to discover that a Radial Basis Function SVM model with a penalty parameter c = 0.3 and a gamma parameter g = 300 produce a model with optimal accuracy that can be trained within a reasonable time frame. We were also able to show that predicting heterozygous genotypes accurately will require further work such as mapping the data to a more linearly separable space before training. Using a 4-fold cross validation technique we were able to show that our model generalizes well to new data and that it did not over-fit our training data set. Lastly, we were able to show that our model has high cohesiveness when predicting non-majority genotype data.

ACKNOWLEDGMENTS

I wanted to thank professor Wei Wang for all her help, guidance, advice and patience during this project. I also wanted to thank Chelsea Ju for working with me on this project as well as answering my many questions regarding this topic.

References

[1] Thermo Fisher Scientific. (2010). TaqMan Gene Expression Assays Protocol. Carlsbad, CA: Life Technologies

[2] Thermo Fisher Scientific. (2018). TaqMan Gene Expression Assayssingle-tube assays User Guide. Pleasanton, CA: Life Technologies

[3] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. Journal of machine learning research, 12(Oct), 2825-2830.

[4] Bengio, Y., & Grandvalet, Y. (2004). No unbiased estimator of the variance of k-fold cross-validation. Journal of machine learning research, 5(Sep), 1089-1105.